Book Title
Book Editors
IOS Press, 2003

Cryptographic Primitives Can Be Fragile

René Peralta [1]
Information Technology Laboratory
National Institute of Standards and Technology

Abstract. We show that a well-known coin-flipping protocol is breakable in the sense that one of the parties can pre-determine the result of the coin-flip. The way in which the protocol fails is illustrative of subtle ways in which a protocol designer may combine secure cryptographic primitives - such as one-way functions and encryption - in a way that produces an *insecure* cryptographic protocol.

Keywords. Cryptographic protocols, coin-flipping, formal verification

1. Introduction

Protocol designers, even experienced ones, sometimes make unwarranted assumptions about cryptographic primitives. Often the same assumptions are made when designing formal verification tools for cryptographic protocols. Whereas the former is an error, the latter is a powerful abstraction tool which can result in the discovery of a protocol flaw. Using abstractions such as nonces, cryptographic signatures, and black-box encryption and decryption, formal methods have been successfully used for finding flaws in some cryptographic protocols. In particular, key-exchange protocols in asynchronous, multi-user environments have been successfully analyzed and debugged using formal methods.

Concrete implementations of cryptographic primitives always have ancillary properties that constrain their use. We illustrate with an example: It is not known whether breaking the RSA cryptosystem is as hard as factoring the modulus N. In fact, there is some evidence that the two problems may not be equivalent [BV98]. A cryptosystem that is provably secure, under the assumption that factoring is hard, is due to Rabin [Rab79]. Rabin's encryption function is simply $F(X) = X^2 \bmod N_A$, where N_A (a product of two large primes) is the public key of A.[2] It is often the case that a sender may wish to send the same message to two recipients A and B. Using Rabin's encryption function, the sender would send $Y_A = M^2 \bmod N_A$ and $Y_B = M^2 \bmod N_B$ over a public channel. It turns out it is easy to recover M from (Y_A, N_A, Y_B, N_B) without having to factor either of the moduli [BW83].

The above problem, caused by an ancillary property of a secure cryptographic primitive, is well known. This note provides examples of protocol failures that are not well

[1]Correspondence to: René Peralta, Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD 20899-8900, U.S.A. Tel.: +1 301 975 2900; Fax: +1 301 840 1357; E-mail: rene.peralta@nist.gov.

[2]Squaring modulo a product of two primes is a four-to-one function. An additional two bits of cyphertext are necessary to fully specify the message being sent. We omit details.

known. In particular, we show that two coin-flipping[3] implementations are breakable in the sense that one of the parties can pre-determine the result of the coin-flip. The way in which the protocols fail is illustrative of the problem being discussed. Namely, there are insecure ways of using secure cryptographic primitives.

Finding the kinds of errors illustrated in this note is currently beyond the capability of formal verification tools. This happens because the problems occur below the level of abstraction of the tools. As formal verification methods become more powerful, they may be able to incorporate "knowledge" about cryptographic primitives into their analyses. We would need to identify and formalize properties that are relevant to security as well as amenable to being incorporated into formal tools. We believe extending the range of applicability of formal methods to the kinds of issues described in this note is an important research problem.

2. One-way functions may not fully hide their argument

A function $f : \mathbb{N} \longrightarrow \mathbb{N}$ is one-way if it is asymptotically hard to invert. Informally, this means the computational complexity of finding an x such that $f(x) = y$ for a given y is super-polynomial in the number of bits of x. Cryptographic protocols typically use *uniformly* hard to invert functions. Informally, this means that if A is a probabilistic-polynomial-time (PPT) algorithm and x is picked uniformly at random among all n-bit numbers, the probability that A, on input $f(x)$, outputs x' such that $f(x) = f(x')$ is super-polynomially small in n. This notion of hardness is quite a bit stronger than the notion of one-way. Consider, for example, the discrete logarithm assumption (DLA):

Let A be a probabilistic polynomial-time algorithm. If

- P is chosen uniformly at random among all n-bit primes;
- g is chosen uniformly at random among all generators of the multiplicative group $\mathbb{Z}_P^*$;
- x is chosen uniformly at random in $\{0, .., P-2\}$;

then the probability that A returns x on input $(P, g, g^x \bmod P)$ is super-polynomially small in n.

The Discrete Logarithm Assumption.

Many useful cryptographic protocols rely on the DLA. Thus, it is important to understand what is not guaranteed by the DLA. In particular, there is nothing in the DLA that says x is completely hidden by $y = g^x \bmod P$. For example, x is even if and only if the Legendre symbol $\left(\frac{y}{P}\right) = y^{\frac{P-1}{2}} \bmod P = 1$. More generally, if d divides $P-1$ then $x \bmod d$ is computable from (g, y, P) in time polynomial in d and $\log P$. In terms of the low-order bits of x, a slightly stronger result shows that, if $P = 2^k T + 1$ with T odd, then the k low-order bits of x can be efficiently computed. Although these facts have been known for decades, the following protocol appears in a standard cryptography reference book (see [Sch96], page 90):

[3] Coin-flipping is an important cryptographic tool due to Manuel Blum [Blu82].

step 0: Alice and Bob agree on a one-way function f and security parameter n.
step 1: Alice chooses uniformly at random an n-bit number x and sends $y = f(x)$ to Bob.
step 2: Bob guesses whether x is odd or even.
step 3: Alice reveals x ; Bob verifies that $f(x)$ is indeed equal to the y he got at step 1; Bob wins the coin-toss if he guessed right at step 2.

Flawed Coin-Flipping Protocol Using One-Way Functions.

The above protocol fails whenever the one-way function f does not hide the least significant bit of its argument (e.g. as in the discrete logarithm). Secure coin-flipping is a well-solved problem (see, for example, [Blu82,CG85,Per86]), and we do not need to "fix" the above protocol. For completeness we do point out that this can be done by

- requiring f to injective and uniformly hard; and
- asking Bob to guess the value of a *hard predicate* of x given $f(x)$.

By "hard predicate" (of x given $f(x)$) we mean a Boolean predicate $h(x)$ such that inverting f is PPT reducible to calculating $h(x)$ on input $f(x)$ for a fraction $\frac{1}{2} + n^{-O(1)}$ of all n-bit x. When $P = 2^k Q + 1$ with Q odd, it is well-known that the $(k+1)^{st}$ least significant bit of x is a hard predicate [Lon84]. Thus, under the discrete logarithm assumption, the following is a way to meet the two requirements above:

- restrict P to primes congruent to 3 modulo 4; and
- ask Bob to guess the *second* least significant bit of x given $(g, g^x \bmod P, P)$.

We also point out there exists a general construction of a hard predicate given an arbitrary uniformly one-way bijection. This result is due to Goldreich and Levin ([GL89]).[4]

3. Subtle (and not so subtle) design mistakes

We now turn to another coin-flipping implementation in the literature (see [Sch96], page 90, "Coin Flipping Using Public-Key Cryptography"). This protocol attempts to use a so-called "commuting" encryption-decryption primitive to implement coin-flipping. Functions f, g are said to commute if $f(g(x)) = g(f(x))$ for all x. RSA encryption and decryption functions are of the form $x^c \bmod N$, and hence commute when the modulus N is constant. That is,

$$\begin{aligned} E_{K1}(D_{K2}(x)) &= (x^{d_2})^{e_1} \bmod N \\ &= (x^{e_1})^{d_2} \bmod N \\ &= D_{K2}(E_{K1}(x)). \end{aligned}$$

[4] A word of caution is warranted here. Suppose we want to flip not one but k coins and we are told $h_1, \ldots, h_k$ are distinct hard predicates. We *cannot* simply use these predicates for our coins, as there is nothing in the definition of hardness that prevents them from being correlated in the sense that knowing some of them may make guessing the others easier. What is needed here is *simultaneous* hardness (see [Per86] for an example of simultaneously hard bits in the discrete logarithm).

The protocol, specialized to RSA and omitting some useless steps, is as follows:

step 0.0: Alice and Bob jointly generate two large primes P, Q and let $N = PQ$.
step 0.A: Alice generates, but keeps secret, e_A and $d_A = e_A^{-1} \bmod \phi(N)$.
step 0.B: Bob generates, but keeps secret, e_B and $d_B = e_B^{-1} \bmod \phi(N)$.
step 1.A: Alice generates two messages m_1, m_2, one indicating heads and the other indicating tails. She sends $\{u_1 = m_1^{e_A} \bmod N, u_2 = m_2^{e_A} \bmod N\}$ to Bob in random order.
step 1.B: Bob chooses one of the u_is at random and sends $v = u_i^{e_B} \bmod N$ to Alice.
step 2.A: Alice sends $w = v^{d_A} = u_i^{e_B d_A} = m_i^{e_A e_B d_A} = m_i^{e_B} \bmod N$ to Bob.
step 2.B: Bob computes $m_i = w^{d_B}$, thereby revealing the result of the coin-flip. He sends m_i to Alice.
step 3.AB: Both Alice and Bob reveal their key pairs and verify the correctness of all previous messages.

Flawed Coin-Flipping Protocol Using Public-Key Cryptography.

The protocol description does not indicate exactly how heads and tails are differentiated. We will see that the protocol is breakable independently of how this is done. Contrary to the situation with the discrete logarithm, the least significant bit of RSA is secure (see [CG85]). Therefore the reader may assume that heads/tails are encoded via the parity of m_i.

Analysis

The author of this protocol provides a "proof of security". Specifically, it is claimed that:

> "This protocol is self-enforcing. Either party can immediately detect cheating by the other, and no trusted third party is required to participate in either the actual protocol or any adjudication after the protocol has been completed."

The proof is flawed on two accounts. First, an assumption about secure encryption is (implicitly) made but turns out to be false. Since we are concerned here with the possibility of formally verifying security, it is instructive to point out this problem. Automated verification tools sometimes make similar assumptions by treating the encryption primitive as a "black box". This may or may not be warranted depending on the specific primitive as well as the way in which it is used. In this case, Schneier states that Alice cannot "read", at step 2.A, the message sent by Bob at step 1.B. Although it is not clear what he means by "read", the subsequent analysis uses this assumption to conclude Alice cannot tell at step 2.A whether Bob sent $v = u_1^{e_B} \bmod N$ or $v = u_2^{e_B} \bmod N$ at step 1.B. This is at least plausible because Alice, despite knowing u_1 and u_2, does not know e_B at this step of the protocol. However, there is nothing in the protocol description to prevent Alice from setting, say, $u_1 = 0$ and $u_2 = 1$. Therefore the assumption is triv-

ially false.[5] However, it is not this trivial attack that concerns us. It is conceivable that restrictions could be put on u_1 and u_2 that would somehow prevent Alice from knowing whether Bob picked u_1 or u_2. For example, Bob could refuse to accept $u_i = m_i^{e_A}$ if he is able to compute m_i. Accordingly, we examine a *much weaker* statement and show that it is also false:

> Suppose Alice and Bob share an RSA modulus N. Let $e \in \mathbb{Z}^*_{\phi(N)}$ be known to Bob but not known to Alice. If Alice sends *uniformly random* x_1, x_2 to Bob, and Bob returns $x_i^e \mod N$ with i chosen by a fair coin, then it is not possible for Alice to tell, with probability significantly better than $\frac{1}{2}$, whether $i = 1$ or $i = 2$.

Flawed Assumption About the Security of RSA.

This does not hold.[6] There are likely many ways for Alice to construct the x_1, x_2 such that she can decide whether i is 1 or 2 by looking at $x_i^e \mod N$. The simplest one is to note that, since e is odd, $x_i^e \mod N$ has the same Jacobi symbol as x_i. Therefore all that Alice has to do is pick the x_1, x_2 with opposite Jacobi symbol.

The second error is not uncommon in the cryptographic literature. In general form, this error is as follows:

- at step i of the protocol, a parameter ω, satisfying certain properties, is supposed to be created by one of the participants;
- at step $i + k$ of the protocol, the parameter ω is disclosed. At this point the other participant checks that ω "was correctly constructed" (meaning it satisfies the properties required by the protocol);
- the "proof" of security then makes the unwarranted assumption that ω was correctly constructed *at step* i.

For example, the fact that the parameters e_A, d_A are revealed at step 3.AB of this protocol does not mean that Alice generated them at step 0.A of the protocol. Furthermore, the fact that Bob can verify the equations $u_i = m_i^{e_A} \mod N$ at step 3.AB does not mean that Alice constructed the u_is in this form at step 1.A.

Having made these observations, the protocol can be broken as follows:

[5] The reader may think this an "unfair" attack on the protocol, as Bob would simply not accept such values. Note, however, that "Bob" is not really a person but a computer program, which will only detect such "aberrant" messages if doing so is explicitly coded.

[6] Of course, even if the assumption did hold, this would not show this protocol is secure (as the protocol does not verify that Alice picks x_1, x_2 uniformly at random). We have worded the assumption in this way because there are cryptographic techniques that can ensure the distribution of the x_is is uniform (e.g. random oracles, strong pseudo-random number generators ran backwards, etc.).

- Alice does not create e_A, d_A at step 0.A.
- At step 1.A Alice sends two random numbers r_1, r_2 with opposite Jacobi symbols.
- Without loss of generality, assume Bob picks r_1 and sends $r_1^{e_B}$ at step 1.B. As explained above, Alice will know that r_1 was chosen because $r_1^{e_B}$ has the same Jacobi symbol as r_1.
- Suppose, without loss of generality, that Alice wants the outcome to be "heads". She picks random ds until r_1^d encodes "heads" and r_2^d encodes "tails".
- Alice sets $d_A = d$, and $e_A = d_A^{-1} \bmod \phi(N)$. At this point the values of m_1, m_2 are "forced" to $m_1 = r_1^{d_A}$ and $m_2 = r_2^{d_A}$ respectively.
- Alice can now complete the protocol without Bob ever knowing he got cheated.

Breaking the Second Coin-Flipping Protocol.

Note that encoding "heads" and "tails" via the parity of m_i is not necessary for our attack to work. The only requirement is that the set of numbers modulo N encoding "heads" and the set encoding "tails" are large enough (and uncorrelated to RSA encryption/decryption) so that a pair of random numbers has a non-negligible chance of encoding different coin-flips. In particular, any known encoding using $O(\log \log N)$ bits of the plaintext is susceptible to this attack.

Acknowledgements

I am indebted to Morris Dworkin for many helpful suggestions on this manuscript.

References

[Blu82] Manuel Blum. Coin flipping by telephone. In *IEEE COMPCON*, pages 133–137, 1982.

[BV98] Dan Boneh and R. Venkatesan. Breaking RSA may not be equivalent to Factoring. In *Proceedings of Advances in Cryptology - Crypto 98*, volume 1403 of *Lecture Notes in Computer Science*, pages 59–71, Santa Barbara, California, USA, 1998. Springer.

[BW83] M. Blum and Alice Wong. personal communication. circa 1983.

[CG85] Benny Chor and Oded Goldreich. RSA/rabin least significant bits are $\frac{1}{2} + 1/$ *poly(log* n*)* secure. In G. R. Blakley and D. C. Chaum, editors, *Advances in Cryptology - Proceedings of CRYPTO 84*, pages 303–313. Springer, 1985. Lecture Notes in Computer Science No. 196.

[GL89] O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. *Proceedings of the 21th Annual ACM Symposium on the Theory of Computing*, pages 25–32, 1989.

[Lon84] D. Long. *The security of bits in the discrete log*. PhD thesis, Princeton University, 1984.

[Per86] R. Peralta. Simultaneous security of bits in the discrete log. In *Advances in Cryptology - Proceedings of EUROCRYPT 85*, Lecture Notes in Computer Science, pages 62–72. Springer-Verlag, 1986.

[Rab79] Michael Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical Report MIT/LCS/TR-212, Laboratory for Computer Science, Massachusetts Institute of Technology, January 1979.

[Sch96] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., second edition, 1996.